LAB # 10

IMPLEMENTATION OF ARRAYS

C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows −

```
type arrayName [ arraySize ];
```

This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement −

```
double balance[10];
```

Initializing Arrays

You can initialize C++ array elements either one by one or using a single statement as follows

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

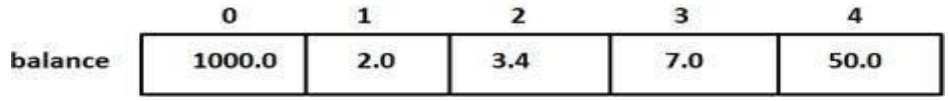
The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array.

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write .

double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};

You will create exactly the same array as you did in the previous example.

balance[4] = 50.0;
The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above –



Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

double salary = balance[9];

The above statement will take 10th element from the array and assign the value to salary variable.

Example

```
#include <iostream>
using namespace std;

int main()
{
   int numbers[5], sum = 0;
   cout << "Enter 5 numbers: ";

   // Storing 5 number entered by user in an array
   // Finding the sum of numbers entered
   for (int i = 0; i < 5; ++i)
   {
      cin >> numbers[i];
      sum += numbers[i];
   }

   cout << "Sum = " << sum << endl;

   return 0;
}
```

Output

Enter 5 numbers: 3
4
5
4
2
Sum = 18

C++ Multi-dimensional Arrays

C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration

type name[size1][size2]...[sizeN];

For example, the following declaration creates a three dimensional 5 . 10 . 4 integer array

int threedim[5][10][4];

Two-Dimensional Arrays

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows –

type arrayName [x][y];

Where type can be any valid C++ data type and arrayName will be a valid C++ identifier. A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array a, which contains three rows and four columns can be shown as below:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form ***a[i][j],*** where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Initializing Two-Dimensional Arrays

Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

```
int a[3][4] =
{
{0, 1, 2, 3} , /* initializers for row indexed by 0 */
{4, 5, 6, 7} , /* initializers for row indexed by 1 */
{8, 9, 10, 11} /* initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example –

```
int val = a[2][3];
```

The above statement will take 4th element from the 3rd row of the array. You can verify it in
the above diagram.

```
#include <iostream>
using namespace std;
int main () {
// an array with 5 rows and 2 columns.
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
// output each array element's value
for ( int i = 0; i < 5; i++ )
for ( int j = 0; j < 2; j++ ){
cout << "a[" << i << "][" << j << "]: ";
cout << a[i][j]<< endl;
}
return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

As explained above, you can have arrays with any number of dimensions, although it is likely that most of the arrays you create will be of one or two dimensions.

Passing Multidimensional Array to a Function

Multidimensional Arrays can be passed to a function as an argument. Consider the following example to pass two-dimensional array to a function:
C++ Program to display the elements of two dimensional array by passing it to a function.

```
#include <iostream>
using namespace std;
void display(int n[3][2]);
int main()
{
int num[3][2] = {
{3, 4},
{9, 5},
{7, 1}
};
display(num);
return 0;
}
void display(int n[3][2])
{
cout << "Displaying Values: " << endl;
for(int i = 0; i < 3; ++i)
{
for(int j = 0; j < 2; ++j)
{
cout << n[i][j] << " ";
}
```

```
}
}
```

Output

Displaying Values:
3 4 9 5 7 1

Lab Tasks

1- Write a program to print elements of an array in reverse order.

2- Write a program that accepts temperature of 7 days from user and print there average using array

3- Write a C++ program to add two matrix by using multi-dimensional arrays.

4- Write a C++ program to find transpose of a matrix by using multi-dimensional arrays.

5- Use two dimensional arrays to store given monthly sale chart of different employees during 1st quarter of year 2021.

	Jan	Feb	Mar	Apr
Emp1	3,20,000	5,56,000	4,98,000	4,79,000
Emp2	5,25,000	4,56,000	5,50,000	4,79,000
Emp3	4,30,000	3,90,000	3,75,000	3,20,000
Emp4	3,25,000	4,59,000	4,55,000	4,95,000
Emp5	4,80,000	5,00,000	4,35,000	4,40,000
Emp6	5,90,000	5,70,000	5,20,000	4,25,000

Now, give function definition only that;

(i) Searches the best employee with highest overall sale during first quarter, so that a letter of appreciation can be awarded to him.

void HighestSale (long [ ] [ ]);

(ii) Takes month number as input, and gives employee number with lowest sale, so that a warning letter can be issued to him to improve his performance.

void PoorPerformance(long [ ] [ ]);